The WebLVC Specification

Reference Manual

VT MÄK
A company of VT Systems

The WebLVC Specification

Reference Manual

VT MÄK
A company of VT Systems

VT MÄK
150 Cambridge Park Drive, 3rd Floor
Cambridge, MA 02140 USA

Voice: 617-876-8085
Fax: 617-876-9208

info@mak.com

www.mak.com

Revision WVC-1.3-3-170213

Contents

Chapter 2. The Standard WebLVC Object Model

Chapter 3. MÄK Extensions to the Standard WebLVC Object Model

Index

Preface

This manual is written for persons who need to write WebLVC applications. The manual assumes that you are familiar with your operating system and windowing environment.

Please see *MÄK WebLVC Server Release Notes* for updates to this manual and the latest information about your version of WebLVC Server.

How the Manual is Organized

This manual is organized as follows:

Chapter 1, *The WebLVC Specification*, introduces the WebLVC specification.

Chapter 2, *The Standard WebLVC Object Model*, introduces the Standard Object Model.

Chapter 3, *MÄK Extensions to the Standard WebLVC Object Model*, describes WebLVC messages that are specifically for use with MÄK applications.

MÄK Products

MÄK WebLVC Server is a member of the VT MÄK line of software products designed to streamline the process of developing and using networked simulated environments. The VT MÄK product line includes the following:

- VR-Link® Network Toolkit. VR-Link is an object-oriented library of C++ functions and definitions that implement the High Level Architecture (HLA) and the Distributed Interactive Simulation (DIS) protocol. VR-Link has built-in support for the RPR FOM and allows you to map to other FOMs. This library minimizes the time and effort required to build and maintain new HLA or DIS-compliant applications, and to integrate such compliance into existing applications.

  VR-Link includes a set of sample debugging applications and their source code. The source code serves as an example of how to use the VR-Link Toolkit to write applications. The executables provide valuable debugging services such as generating a predictable stream of HLA or DIS messages, and displaying the contents of messages transmitted on the network.

- MÄK RTI. An RTI (Run-Time Infrastructure) is required to run applications using the High Level Architecture (HLA). The MÄK RTI is optimized for high performance. It has an API, RTIspy®, that allows you to extend the RTI using plug-in modules. It also has a graphical user interface (the RTI Assistant) that helps users with configuration tasks and managing federates and federations.
- VR-Forces®. VR-Forces is a computer generated forces application and toolkit. It provides an application with a GUI, that gives you a 2D and 3D views of a simulated environment.

  You can create and view entities, aggregate them into hierarchical units, assign tasks, set state parameters, and create plans that have tasks, set statements, and conditional statements. You can simulate using entity-level modeling, which focuses on the actions of individual people and vehicles, and aggregate-level modeling, which focuses on the interaction of large hierarchical units.

  VR-Forces also functions as a plan view display for viewing remote simulation objects taking part in an exercise. Using the toolkit, you can extend the VR-Forces application or create your own application for use with another user interface.

- VR-Vantage™. VR-Vantage is a line of products designed to meet your simulation visualization needs. It includes three end-user applications (VR-Vantage Stealth, VR-Vantage PVD, and VR-Vantage IG) and the VR-Vantage Toolkit.
  - VR-Vantage Stealth displays a realistic, 3D view of your virtual world, a 2D plan view, and an exaggerated reality (XR) view. Together these views provide both situational awareness and the big picture of the simulated world. You can move your viewpoint to any location in the 3D world and can attach it to simulation objects so that it moves as they do.
  - VR-Vantage IG is a configurable desktop image generator (IG) for out the window (OTW) scenes and remote camera views. It has most of the features of the Stealth, but is optimized for its IG function.
  - VR-Vantage PVD provides a 2D plan view display. It gives you the big picture of the simulated world.
  - SensorFX. SensorFX is an enhanced version of VR-Vantage that uses physics based sensors to view terrain and simulation object models that have been materially classified. It is built in partnership with JRM Technologies.
  - The VR-Vantage Toolkit is a 3D visual application development toolkit. Use it to customize or extend MÄK's VR-Vantage applications, or to integrate VR-Vantage capabilities into your custom applications. VR-Vantage is built on top of OpenSceneGraph (OSG). The toolkit includes the OSG version used to build VR-Vantage.
- MÄK Data Logger. The Data Logger, also called the Logger, can record HLA and DIS exercises and play them back for after-action review. You can play a recorded file at speeds above or below normal and can quickly jump to areas of interest. The Logger has a GUI and a text interface. The Logger API allows you to extend the Logger using plug-in modules or embed the Logger into your own application. The Logger editing features let you merge, trim, and offset Logger recordings.
- VR-Exchange™. VR-Exchange allows simulations that use incompatible communications protocols to interoperate. For example, within the HLA world, using VR-Exchange, federations using the HLA RPR FOM 1.0 can interoperate with simulations using RPR FOM 2.0, or federations using different RTIs can interoperate. VR-Exchange supports HLA, TENA, and DIS translation.
- VR-TheWorld™ Server. VR-TheWorld Server is a simple, yet powerful, web-based streaming terrain server, developed in conjunction with Pelican Mapping. Delivered with a global base map, you can also easily populate it with your own custom source data through a web-based interface. The server can be deployed on private, classified networks to provide streaming terrain data to a variety of simulation and visualization applications behind your firewall.

- DI-Guy™. The DI-Guy product line is a set of software tools for real-time human visualization, simulation, and artificial intelligence. Every DI-Guy software offering comes with thousands of ready-to-use characters, appearances, and motions. DI-Guy enables the easy creation of crowds and individuals who are terrain aware, autonomous, and react intelligently to ongoing events. Save time, money and create outstanding simulations with DI-Guy. The DI-Guy product line includes the following products:
  - The DI-Guy SDK. Embed the DI-Guy library in your real-time application and populate your world with lifelike human characters.
  - DI-Guy Scenario™. Author and visualize human performances in a rich, user-friendly graphical environment. Use DI-Guy Scenario as an end visualization application or save scenarios and load them into your DI-Guy SDK enabled application.
  - ECOSim. Enhanced Company Operations Simulation (ECOSim) is a company-level training simulation that teaches leaders how best to deploy troops, UAVs, convoys, and other assets. ECOSim focuses on ease-of-use, rapid scenario generation, runtime operator control, and realistic and reactive human simulation.
  - DI-Guy AI. Generate crowds of autonomous characters to quickly populate your worlds with hundreds and thousands of terrain-aware, collision avoiding DI-Guys. Used as a module on top of DI-Guy Scenario and DI-Guy SDK.
  - Expressive Faces Module. Enable DI-Guy characters to have faces that display emotion, eyes that look in directions and blink, and lips that sync to sound files.
  - DI-Guy Motion Editor. Create or customize motions to your particular needs in an easy-to-use graphical application.
- RadarFX. RadarFX is a client-server application that simulates synthetic-aperture radar (SAR). The server application, which is based on VR-Vantage and SensorFX, loads a terrain database and, optionally, connects to simulations. A client application requests SAR images from the server. RadarFX includes a sample client application.
- VR-Engage. VR-Engage is a multi-role virtual simulator that lets users play the role of a first person human character, a ground vehicle driver, gunner or commander, or the pilot of a fixed wing aircraft or helicopter. It incorporates the VR-Force simulation engine and the VR-Vantage graphics rendering capabilities.
- WebLVC Server. WebLVC Server implements the server side of the WebLVC protocol so that web-based simulation federates can participate in a distributed simulation. It is part of the WebLVC Suite, which includes the server and several sample applications that work with VR-Forces and VR-Vantage.

How to Contact Us

For WebLVC Server technical support, information about upgrades, and information about other MÄK products, you can contact us in the following ways:

Telephone

Call or fax us at:	Voice:	617-876-8085 (extension 3 for support)
	Fax:	617-876-9208

E-mail

Sales and upgrade information:	info@mak.com
Technical support:	support@mak.com

Internet

MÄK web site home page:	www.mak.com
License key requests:	www.mak.com/support/licenses/get-licenses
Product version and platform information:	www.mak.com/support/product-versions
For the free, unlicensed MÄK RTI:	www.mak.com/resources/bonus-material
MÄK Community Forum:	www.mak.com/connect/forum

Post

Send postal correspondence to:	VT MÄK 150 Cambridge Park Drive, 3rd Floor Cambridge, MA, USA 02140

When requesting support, please tell us the product you are using, the version, and the platform on which you are running.

Document Conventions

This manual uses the following typographic conventions:

`Monospaced`	Indicates commands or values you enter.
`Monospaced Bold`	Indicates a key on the keyboard.
Monospaced Italic	Indicates command variables that you replace with appropriate values.
Blue text	A hypertext link to another location in this manual or another manual in the documentation set.
`{ }`	Indicates required arguments.
`[ ]`	Indicates optional arguments.
|	Separates options in a command where only one option may be chosen at a time.
`(	)`	In command syntax, indicates equivalent alternatives for a command-line option, for example, `(-h | --help)`.
/	Indicates a directory. Since MÄK products run on both Linux and Windows PC platforms, we use the / (slash) for generic discussions of pathnames. If you are running on a PC, substitute a \ (backslash) when you type pathnames.
Italic	Indicates a file name, pathname, or a class name.
sans Serif	Indicates a parameter or argument.
➤	Indicates a one-step procedure.
Menu → Option	Indicates a menu choice. For example, an instruction to select the Save option from the File menu appears as: Choose **File → Save**.
	Click the icon to run a tutorial video in the default browser.
	Indicates supplemental or clarifying information.
	Indicates additional information that you must observe to ensure the success of a procedure or other task.

Directory names are preceded with dot and slash characters that show their position with respect to the WebLVC Server home directory. For example, the directory *logger1.3/doc* appears in the text as *./doc*.

Mouse Button Naming Conventions

An instruction to click the mouse button, refers to clicking the primary mouse button, usually the left button for right-handed mice and the right button for left-handed mice. The context-sensitive menu, also called a popup menu or right-click menu, refers to the menu displayed when you click the secondary mouse button, usually the right button on right-handed mice and the left button on left-handed mice.

Third Party Licenses

MÄK software products may use code from third parties. This section contains the license documentation required by these third parties.

Boost License

VR-Link, and all MÄK software that uses VR-Link uses some code which is distributed under the Boost License. All header files that contain Boost code are properly attributed. The Boost web site is: www.boost.org.

Boost Software License - Version 1.0 - August 17th, 2003

Permission is hereby granted, free of charge, to any person or organization obtaining a copy of the software and accompanying documentation covered by this license (the "Software") to use, reproduce, display, distribute, execute, and transmit the Software, and to prepare derivative works of the Software, and to permit third-parties to whom the Software is furnished to do so, all subject to the following:

The copyright notices in the Software and this entire statement, including the above license grant, this restriction and the following disclaimer, must be included in all copies of the Software, in whole or in part, and all derivative works of the Software, unless such copies or derivative works are solely in the form of machine-executable object code generated by a source language processor.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT. IN NO EVENT SHALL THE COPYRIGHT HOLDERS OR ANYONE DISTRIBUTING THE SOFTWARE BE LIABLE FOR ANY DAMAGES OR OTHER LIABILITY, WHETHER IN CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

libXML and libICONV

VR-Link and all MÄK software that uses VR-Link, links in libXML and libICONV. On some platforms the compiled libraries and header files are distributed with MÄK Products. MÄK has made no modifications to these libraries. For more information about these libraries please see the following web sites:

- The LGPL license is available at: http://www.gnu.org/licenses/lgpl.html
- Information about IconV is at: http://www.gnu.org/software/libiconv/
- Information about LibXML is at: http://xmlsoft.org/

pThreads Library

VR-Exchange links with the pThreads win32 library. The library is distributed under the GNU Lesser General Public License (LGPL). MÄK has made no modification to this library. For information about the pThreads win32 library please see: http://source-ware.org/pthreads-win32/

EHS, PCRE, and PME

The MÄK RTI links with the EHS, PCRE, and PME libraries. These libraries are distributed under the GNU Lesser General Public License (LGPL). MÄK has made modification to these libraries only to allow them to compile on the supported plat-forms. For more information about these libraries please see the following web sites:

- EHS: http://xaxxon.slackworks.com/ehs/
- PCRE: http://www.pcre.org/
- PME: http://xaxxon.slackworks.com/pme/index.html

1. The WebLVC Specification

This chapter is an introduction to the WebLVC specification as implemented in MÄK WebLVC Server.

1.1. Introduction to WebLVC

WebLVC is an interoperability protocol that enables web-based applications (typically JavaScript applications running in a web browser) to interoperate in modeling and simulation (M&S) federations. WebLVC client applications communicate with the rest of the federation through a WebLVC server, which participates in the federation on behalf of one or more clients. The WebLVC protocol defines a standard way of passing simulation data between a web-based client application and a WebLVC server, independent of the protocol used in the federation. Thus, a WebLVC client can participate in a DIS exercise, an HLA federation, a TENA execution, or other distributed simulation environment.

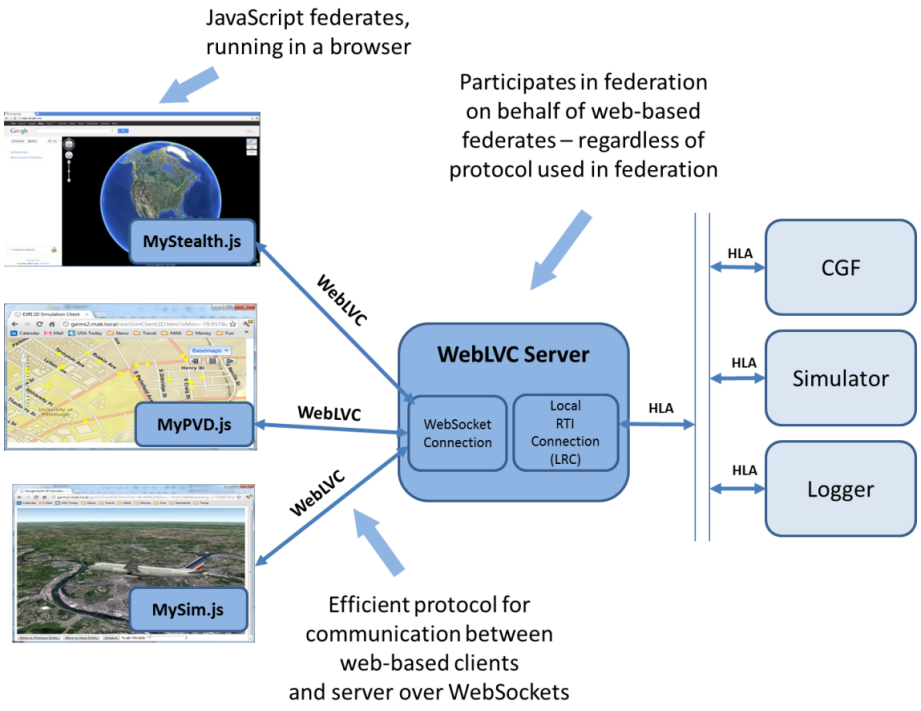

Figure 1-1. WebLVC protocol

The WebLVC protocol specifies a standard way of encoding object update messages, interaction messages, and administrative messages as JSON (JavaScript Object Notation) objects, which are passed between client and server using WebSockets. WebLVC is flexible enough to support representation of arbitrary types of objects and interactions (that is, arbitrary object models). However, WebLVC includes a Standard Object Model definition based on the semantics of the DIS protocol and HLA's RPR FOM. Users can extend the Standard Object Model by adding new types of objects, attributes, interactions, and parameters, or can choose to represent the semantics of entirely different object models (for example, other HLA FOMs, ANDEM models, and so on.)

1.1.1. Introduction to JSON

A JSON object is a collection of unordered name/value pairs, expressed in text, using the syntax of JavaScript. For example, a JSON object representing a can of soda might look like this:

```
{
  “ContainerType” : “can”,
  “SizeInLiters” : 0.33,
  “Brand” : “Coca Cola”,
  “Flavor” : “Cherry”
}
```

JSON is the natural way of encoding structured data that is intended to be used by web-based JavaScript federates, because a JSON object is a string-literal representation of a JavaScript object. The JavaScript language has built-in serialize and parse functions that generate JSON strings from JavaScript objects, and vice versa, without the need for data marshaling or conversion.

Upon receiving the soda can object described above, a JavaScript federate can directly query the object for its attribute values:

```
var numLiters = sodaCan.SizeInLiters;
```

1.1.2. A WebLVC Example

The WebLVC specification defines a standard mechanism for representing simulation interoperability messages as JSON objects by specifying property names, data types, enumeration values, and conventions. For example, in order to represent a valid WebLVC attribute object message, a JSON object must include a property called MessageKind, a property called ObjectType, and a variety of other properties defined by the object model being used and the object type of the object being updated. The WebLVC Standard Object Model uses attributes names, types, and enumerated values dictated by DIS, the RPR FOM, and the SISO Enumerations Document.

For example, a web-based flight simulator using WebLVC with the Standard Object Model might convey the current state of the aircraft it is simulating by sending the following WebLVC message:

```
{
  MessageKind : 1,
  ObjectName : “F-16 Alpha”,
  ObjectType : “WebLVC:PhysicalEntity”,
  EntityIdentifier : [1,2,1],
  EntityType : [1,2,225,1,3,0,0],
  WorldLocation : [4437182.0232, -395338.0731, 873923.4663],
  VelocityVector : [57.04, 32.77, 89.263],
  Orientation : [-1.65, 2.234, -0.771],
  Marking : “F-16”,
  DamageState : 0
}
```

A MessageKind of 1 indicates that this message is conveying an AttributeUpdate. F-16 Alpha is the name of the WebLVC Object being updated, and the object is of type WebLVC:PhysicalEntity – one of the types in the Standard WebLVC Object Model.

- The various attribute names match those in the RPR FOM.
- The DIS and RPR FOM (DIS/RPR) style EntityIdentifier and EntityType structures are encoded as arrays of enumerated values.
- The same units and coordinate systems as DIS/RPR are used for WorldLocation, VelocityVector, and Orientation.
- Any given WebLVC AttributeUpdate may contain values for only a subset of the object's properties, for example, values that have changed since the last update.

1.2. Basic Protocol Specification

This section defines the basic, object-model-independent elements of the WebLVC protocol. It includes specifications for the following:

- Header information that is common to all WebLVC messages.
- Administrative messages, such as Connect and ObjectDeletion messages, which do not depend on the object model being used.
- Common properties, such as ObjectName and TimeStamp that may be included in Attribute Update and Interaction messages regardless of what kinds of objects and interactions they are describing.

1.2.1. Header Information

Since WebLVC messages are simply JSON objects, there is no concept of a "message header" per se. Header information, such as the type of message a JSON object represents, is encoded just like the rest of the data in a message – through named properties and values.

Required Properties

MessageKind. All WebLVC messages must include the MessageKind property. MessageKind is a number that indicates what kind of WebLVC message an object represents, as follows:

- 0 – Other.
- 1 – AttributeUpdate message (updates the values of attributes of a WebLVC object).
- 2 – Interaction message (describes the parameters of an event or interaction).
- 3 – Connect message (simple handshake to connect client to server).
- 4 – ObjectDeletion message (informs recipient that a WebLVC object no longer exists).

A client or server may check the value of the MessageKind property to determine what kind of message it has received, and therefore determine what additional properties it can expect the message to contain.

1.2.2. The Connect Message

A client application is not required to send a Connect message to the server in order to connect. The client can start sending AttributeUpdates and Interactions immediately upon opening its WebSocket to the server. However, it is often desirable to provide the server with additional information about the client, so that the server can pass such information along to the rest of the federation.

Required Properties

ClientName. ClientName is a string representing the name of the client federate.

Optional Properties

WebLVCVersion. A number representing the version of WebLVC to be used for communication between the connecting client and server. If WebLVCVersion is omitted, it is assumed that the client is using the default version of the WebLVC standard supported by the server.

Example Connect Message

```
{
  MessageKind : 3,
  ClientName : “MyFederate”,
  WebLVCVersion : 0.1
}
```

1.2.3. The ObjectDeletion Message

An application must send an ObjectDeletion message to inform the recipient that an object it has been updating no longer exists.

Required Properties

ObjectName. ObjectName is a string representing the name of the object that has been deleted.

Example ObjectDeletion Message

```
{
  MessageKind : 4,
  ObjectName : “F-16 Alpha”,
}
```

There is no corresponding ObjectCreation or ObjectRegistration message defined in WebLVC. This is because the first AttributeUpdate sent for an object can serve to inform the recipient that an object exists with the specified name. (This is similar to the DIS protocol, where a recipient is informed that a particular simulation object exists when it receives its first Entity State PDU from a previously-unknown simulation object.)

1.2.4. AttributeUpdate Message

AttributeUpdate messages are used to convey values of the attributes of a WebLVC object. While the names of the properties that carry the object's state depend on the object model, there are several common properties, such as ObjectName and TimeStamp, whose definitions are independent of the type of object being updated. The rules that dictate when, or how frequently, an AttributeUpdate shall be sent depend on the Object Model being used, and the type of object being updated.

Required Properties

ObjectName. The value of the ObjectName property is a string that uniquely identifies the WebLVC object for which the message is conveying an update. It is the responsibility of the application that is sending updates for an object (which might be either a client or the WebLVC Server) to ensure that it chooses a name that is not already in use.

Optional Properties

- ObjectType. A string that indicates the Object Type of the WebLVC object for which the message is conveying an update (similar to the concept of HLA Object-Class or DIS PDU Kind). Based on the value of the ObjectType property, an application can determine what other properties it can expect the message to contain. For example, updates for objects of type WebLVC:PhysicalEntity may contain a WorldLocation property.

  Strings are used to represent object types (rather than enumerated numbers), to support easy extensibility. Strings allow naming conventions that make collisions unlikely among different users' extensions and avoid the need for a central repository of enumeration-to-string mappings for user extensions. By convention, object types defined by the Standard WebLVC Object Model have the prefix WebLVC: (for example, WebLVC:PhysicalEntity), while user extensions have a prefix that indicates the origin of an object model (for example, MyProject:MyObjectKind or MyCompany:MyObjectKind).

  Although the ObjectType property is quite important, it is not considered to be required, because it need not be included in every WebLVC message. The first AttributeUpdate for each WebLVC object must include the ObjectType property to inform the receiving application of the object type associated with each WebLVC object. The receiving application can then store that association, so that subsequent AttributeUpdate messages do not need to include the ObjectType property. As is the case with all optional properties, an application may choose to include ObjectType in every AttributeUpdate (for example, to relieve the receiver of the responsibility to look up the type of each object by name in order to know how to interpret the update).

- Timestamp. The time at which the data is valid, using the DIS timestamp field format converted into hexadecimal ASCII character representation as defined by the RPR FOM. If Timestamp is missing, the message is assumed to be valid at the time of receipt.

1.2.5. Interaction Message

Interaction messages send parameters of a simulation event or interaction. While the names of the properties that carry the interactions parameters depend on the object model, there are several common properties, such as TimeStamp, whose definitions are independent of the type of interaction being sent. The rules that dictate when, or how frequently, an interaction must be sent depend on the object model being used and the type of interaction in question.

Required Properties

InteractionType. A string that indicates the type of the interaction sent by a WebLVC message (similar to HLA InteractionClass or DIS PDU Kind). All interaction messages must include the InteractionType property. Based on the value of the InteractionType property, an application can determine what other properties it can expect the message to contain. For example, interactions of type WebLVC:WeaponFire may contain a MunitionType property.

Strings are used to represent interaction types (rather than enumerated numbers), to support easy extensibility. Strings allow naming conventions that make collisions unlikely among different users' extensions, and avoid the need for a central repository of enumeration-to-string mappings for user extensions. By convention, interaction types defined by the Standard WebLVC Object Model have the prefix WebLVC: (for example, WebLVC:WeaponFire), while user extensions have a prefix that indicates the origin of an object model (for example, MyProject:MyInteractionKind or MyCompany:MyInteractionKind).

Optional Properties

Timestamp. The time at which the data is valid, using the DIS timestamp field format converted into hexadecimal ASCII character representation as defined by the RPR FOM.

1.3. Extending the WebLVC Protocol

The WebLVC standard includes a Standard Object Model based on a subset of the DIS/RPR object model. However, the WebLVC standard can be easily extended to facilitate the exchange of data that is outside of the Standard Object Model.

- You can define new ObjectTypes and InteractionTypes, and define various properties associated with each new message type.
- You can add new properties to existing StandardObjectModel message types.

JSON (and Javascript) is naturally extensible in a backwards compatible way. When a client receives a message, it can check the message type against types it knows about, and ignore messages of unknown types. Similarly, when a client receives a message of a type is knows about, it can query the message for values for properties it expects, and ignore properties whose names are unexpected.

1.3.1. Extending WebLVC Automatically, Based on DIS

Imagine that a DIS application has extended the DIS protocol (or is using some of the built-in extension capabilities of DIS 7) to add new attributes to the Entity State PDU. If you want to pass that attribute to a WebLVC client, you can define a WebLVC (JSON) representation of the attribute, and define a new property of WebLVC:PhysicalEntity to convey it.

There is no formal meta-model in WebLVC to represent object model extensions – you can document extensions in a way similar to the way this manual defines the standard WebLVC Object Model. To implement the use case in the previous paragraph, extend your WebLVC server to support the mapping of the new DIS attribute to the new WebLVC property. Depending on how the WebLVC server is built and distributed, this extension might be accomplished through editing source code, using a plug-in API, or through a run-time configuration mechanism. However it is accomplished, the point is that user intervention is required in order to "teach" the WebLVC server which DIS field or new PDU type should be mapped to which WebLVC property or message type.

1.3.2. Extending WebLVC Automatically, Based on an HLA FOM

If you have defined an object model in the form of an HLA FOM, a WebLVC server can automatically extend the WebLVC protocol to include WebLVC elements that directly correspond to custom HLA Object Classes, Interaction Classes, Attributes and Parameters. This is because unlike typical DIS extensions, HLA FOM elements are machine readable in a standard way. A FOM provides information about class names, as well as attribute and parameter names and datatypes – everything that an application needs to know how to build corresponding JSON elements within a WebLVC object model.

Not only is it possible for a WebLVC Server to extend the WebLVC protocol, it is also possible for a WebLVC server to automatically know how to translate between the new FOM elements and the corresponding new WebLVC elements – without requiring user extension of server capabilities. In order to achieve this, there must be a well-documented set of rules for mapping between HLA data types and structures and their corresponding WebLVC representations. If all WebLVC servers map a specific HLA data type to the same representation on the WebLVC side, client developers will know exactly what to expect, and can write client code accordingly.

Table 1-1 lists data types used by the JSON format.

Table 1-1: JSON type examples

JSON type	Examples
Number	3.14
String	"my name"
Boolean	true, false
Array	[1, 2, "3"]

Table 1-1: JSON type examples

JSON type	Examples
Object	{ a: 1, b: “a string” }
null	Null

Because JSON has only a few types, there might be many valid mappings from simulation data models with a greater set of more specific types. Should a WebLVC number be mapped as an integer or some floating point type? Should a JSON string map as an array of characters, or does it correspond to some enumerated value? Starting from the more specific data model should simplify the conversion.

1.3.3. Standard Mapping Rules for HLA and the RPR FOM

The High-Level Architecture uses a meta-model called an object model template (OMT) to encode simulation data into a Federation Object Model (FOM). The OMT organizes data type definitions into tables. Rules and recommendation for mapping between standard HLA and RPR FOM datatypes, and WebLVC datatypes are described in the following sections.

The RPR FOM includes datatypes intended only as padding, to ensure correct alignment of data. These padding datatypes do not map to corresponding WebLVC datatypes and are omitted from WebLVC datatypes.

Basic Data Representation Table

The OMT describes representations of data separately from their types. Unless a more specific mapping applies, OMT simple datatypes should use the mapping of their representation. Basic data representations probably map as either Number, String or Boolean data types.

Table 1-2 lists the basic data representations for HLA.

Table 1-2: HLA basic data representation

OMT name	WebLVC type
HLAinteger16BE	Number
HLAinteger32BE	Number
HLAinteger64BE	Number
HLAfloat32BE	Number
HLAfloat64BE	Number
HLAoctetPairBE	Number (unsigned)
HLAinteger16LE	Number
HLAinteger32LE	Number
HLAinteger64LE	Number
HLAfloat32LE	Number
HLAfloat64LE	Number
HLAoctetPairLE	Number (unsigned)
HLAoctet	Number (unsigned)
RPRunsignedInteger8BE	Number (unsigned)
RPRunsignedInteger16BE	Number (unsigned)
RPRunsignedInteger32BE	Number (unsigned)
RPRunsignedInteger64BE	Number (unsigned)
Unsignedinteger16BE	Number (unsigned)
Unsignedinteger32BE	Number (unsigned)
Unsignedinteger64BE	Number (unsigned)
OMT13any	Number (unsigned)

Simple Datatype Table

OMT character datatypes should map as single character strings.

Table 1-3 lists scalar data types.

Table 1-3: HLA simple datatype table

OMT name	WebLVC type
HLAASCIIchar	String (single character)
HLAunicodeChar	String (single character)
HLAbyte	Number (unsigned)
HLAinteger64Time	Number
HLAfloat64Time	Number
HLAfederateHandle	Number
HLAmsec	Number
HLAseconds	Number
HLAcString	String
HLAstring	String
HLAcount	String
handle	String
timeType	String
string	String
int	Number
long	Number
long long	Number
octet	Number (unsigned)
short short	Number
unsigned int	Number (unsigned)
unsigned long	Number (unsigned)
unsigned long long	Number (unsigned)
unsigned short	Number (unsigned)
double	Number
float	Number

Enumerated Datatype Table

Table 1-4 lists enumerated datatypes.

Table 1-4: HLA enumerated datatype table

OMT name	OMT Enumerator	WebLVC type	WebLVC value
HLAboolean	HLAfalse	Boolean	false
	HLAtrue	Boolean	true
RPRboolean	HLAfalse	Boolean	false
	HLAtrue	Boolean	true
OMT13boolean	HLAfalse	Boolean	false
	HLAtrue	Boolean	true

Except for the Boolean enumerations explicitly listed in Table 1-4, map the values of enumerators as numbers. Using the enumerator name as a string would require updating intermediating software (for example, gateways) as the strings change. Numbers can be passed without translation, so new enumerations names are ignored.

Table 1-5 is an example of enumerated datatypes for HLA.

Table 1-5: Example HLA enumerated datatype table

OMT name	OMT Enumerator	OMT value	WebLVC type	WebLVC value
HLAboolean	HLAfalse	0	Boolean	false
	HLAtrue	1	Boolean	true
AntennaPattern-TypeEnum32	OmniDirectional	0	Number	0
	Beam	1	Number	1
	SphericalHar-monic	2	Number	2

Array Datatype Table

Standard HLA arrays are converted to either strings or arrays as shown in Table 1-6.

Table 1-6: HLA array datatype table

OMT name	WebLVC type
HLAASCIIstring	String
HLAunicodeString	String
HLAopaqueData	Array (of unsigned Number)
HLAtoken	n/a
MarkingArray11	String
MarkingArray31	String
OctetArray {all variations}	Array (of unsigned Number)

HLAtoken is a zero-length array and is not converted to a WebLVC representation because JSON already has termination rules for strings and arrays.

WebLVC does not distinguish between fixed and variable length arrays.

OMT arrays should convert to WebLVC arrays, except character arrays (where array element representations map as characters, or octets intended as characters). Character arrays should convert to WebLVC strings.

Fixed Record Datatype Table

Fixed records are mapped as JSON objects. Each field in the OMT is mapped to a JSON object name/value pair, where the name is the OMT field name and the value depends on the OMT field type.

Table 1-7: Example HLA fixed record datatype table

Record name	Field			Encoding	Semantics
	Name	Type	Semantics		
MessageLocation	X	HLAfloat64BE	X coordinate of message origination.	HLAfixedRecord	Coordinates where the message originates.
	Y	HLAfloat64BE	Y coordinate of message origination.		

A fixed record datatype would map as:

```
{
  "X" : 2.78,
   "Y" : 3.14
}
```

Variant Record Datatype Table

Variant records are mapped as an array of length 2. The first element is the enumerator in string representation. The second element is the value of the corresponding alternative. Table 1-8 and Table 1-9 are an example of the information in a variant record datatype table. (The information is broken up into two tables to fit on the page.)

Table 1-8: Example HLA variant record datatype table

Record Name	Encoding	Semantics
Waiter Value	HLAvariantRecord	Datatype for waiter performance rating value.

Table 1-9: Example discriminant and alternative for Waiter Value record

Discriminant				Alternative		
Name	Type	Enumerator	Value	Name	Type	Semantics
Val Index	Experience Level	Trainee	0	Course Passed	HLAboolean	Ratings scale for employees under training.
		[Apprentice … Senior], Master	1	Rating	RateScale	Ratings scale for permanent employees.
		HLAother	2	NA	NA	All others.

Possible mappings:

- [0, true] – a trainee that has passed the course.
- [1, 8.0] – depends on the definition of RateScale.

For a more comprehensive example, Table 1-10 defines the RPR FOM fixed record SpatialStruct (semantics omitted).

Table 1-10: SpatialStruct definition

Record name	Field		Encoding
	Name	Type	
SpatialStruct	DeadReckoningAlgorithm-A-Alternatives	SpatialStruct-DeadReckoningAlgorithm	HLAfixedRecord

It contains a single field DeadReckoningAlgorithm-A-Alternatives which is the variant record SpatialStruct-DeadReckoningAlgorithm (semantics omitted).

Table 1-11: SpatialStruct-DeadReckoningAlgorithm definition

Record Name	Discriminant				Alternative		Encoding
	Name	Type	Enumerator	Value	Name	Type	
SpatialStruct-DeadReckoningAlgorithm	DeadReckoningAlgorithm	DeadReckoningAlgorithmEnum8	Other	0			HLAvariantRecord
			Static	1	SpatialStatic	SpatialStatic-Struct	
			DRM_FPW	2	SpatialFPW	SpatialFPStruct	
			DRM_RPW	3	SpatialRPW	SpatialRPStruct	
			DRM_RVW	4	SpatialRVW	SpatialRVStruct	
			DRM_FVW	5	SpatialFVW	SpatialFVStruct	
			DRM_FPB	6	SpatialFPB	SpatialFPStruct	
			DRM_RPB	7	SpatialRPB	SpatialRPStruct	
			DRM_RVB	8	SpatialRVB	SpatialRVStruct	
			DRM_FVB	9	SpatialFVB	SpatialFVStruct	

The fixed record SpatialStruct would be mapped to a JSON object containing a single property named DeadReckoningAlgorithm-A-Alternatives. That property would be an array of length 2, corresponding to the variant record SpatialStruct-DeadReckoningAlgorithm.

As a further example, assuming the discriminant enumerator was DRM_FVB, then the alternative would be of type SpatialFVStruct. The required definitions are shown in Table 1-12.

Table 1-12: SpatialFVStruct and dependencies definitions

Record name	Field		Encoding
	Name	Type	
SpatialFVStruct	WorldLocation	WorldLocationStruct	HLAfixedRecord
	IsFrozen	OMT13boolean	
	Orientation	OrientationStruct	
	VelocityVector	VelocityVectorStruct	
	AccelerationVector	AccelerationVector-Struct	
WorldLocation-Struct	X	Double	HLAfixedRecord
	Y	Double	
	X	Double	
OrientationStruct	Psi	Float	HLAfixedRecord
	Theta	Float	
	Phi	Float	
VelocityVector-Struct	XVelocity	Float	HLAfixedRecord
	YVelocity	Float	
	ZVelocity	Float	
AccelerationVec-torStruct	XAcceleration	Float	HLAfixedRecord
	YAcceleration	Float	
	ZAcceleration	Float	

OMT13boolean would be mapped as WebLVC boolean because the enumerators are part of the HLA specification (they begin with the HLA prefix). Otherwise they would be mapped as their numeric values.

Table 1-13: OMT13boolean definition

OMT name	OMT Enumerator	OMT value	WebLVC type	WebLVC value
OMT13boolean	HLAfalse	0	Boolean	false
	HLAtrue	1	Boolean	true

The following full example has comments after the slashes (//).

```
{ // SpatialStruct object
  “DeadReckoningAlgorithm-A-Alternatives” : // only attribute
    [9, // discriminant DRM_FVB converted as number
     { // SpatialFVStruct converted as object
      “WorldLocationStruct”: {
        “X”: 4437182.0232,
        “Y”: -395338.0731,
        “Z”: 873923.4663 },
      “IsFrozen”: false,
      “OrientationStruct”: {
        “Psi”: -1.65,
      “Theta”: 2.234
        “Phi”: -0.771 },
      “VelocityVectorStruct”: {
        “X”: 57.04,
      “Y”: 32.77,
        “Z”: 89.263 },
      “AccelerationVectorStruct”: {
        “XAcceleration”: 0.0,
        “YAcceleration”: 0.0,
        “ZAcceleration”: 0.0 }
     } // end of SpatialFVStruct object
  ] // end of DeadReckoningAlgorithm-A-Alternatives array
} // end of SpatialStruct object
```

Assume spatialObj contains the above SpatialStruct object. Access to the X position would look like:

```
spatialObj.DeadReckoningAlgorithm-A-Alternatives[1].
  WorldLocationStruct.X
```

For comparison, consider access to the X position of a WebLVC Coordinates object coordObj:

```
coordObj.WorldLocation[0]
```

2. The Standard WebLVC Object Model

This document is an introduction to the WebLVC specification as implemented in MÄK WebLVC Server.

2.1. The Standard WebLVC Object Model

The WebLVC specification is flexible enough to allow the representation of arbitrary object models by defining appropriate object types, interaction types, and appropriate properties of specific kinds of AttributeUpdate messages and Interaction messages.

WebLVC includes a Standard Object Model, which was developed to serve the needs of web-based federates that require semantics similar to those in the DIS protocol or HLA's RPR FOM. The Standard WebLVC Object defines a set of object types, interaction types, and corresponding AttributeUpdate and Interaction messages based on the DIS/RPR definitions. Wherever possible, the Standard Object Model uses attribute names, parameter names, data types, and enumerations that match the corresponding elements in DIS/RPR.

Although in general the WebLVC Object Model mirrors DIS/RPR very closely, there is not always a direct mapping between RPR FOM classes, attributes, and parameters and their WebLVC counterparts. This is because the WebLVC specification attempts to represent data in a way that is consistent with JSON conventions, efficient to convey in JSON messages, and easy to parse and use in JavaScript applications.[1] For example, the EntityType property of a PhysicalEntity object uses the familiar 7-element SISO enumeration, but it encodes that 7-tuple as a simple JSON array of 7 numbers – for example, [1,1,225,1,1,0,0] – rather than using the 8-byte binary "struct" used in DIS and the RPR FOM. As another example, WebLVC uses a single WebLVC:PhysicalEntity type regardless of which kind of simulation object is being simulated, rather than the RPR FOM's fairly deep hierarchy of subclasses of the PhysicalEntity class. (Many of the reasons for choosing a deep hierarchy in the RPR FOM do not apply in WebLVC, where there is no real concept of classes, inherited attributes, object promotion, and so on.)

That being said, the similarities between the WebLVC Object Model and the RPR FOM are far more numerous than the differences.

The object types defined by the Standard WebLVC Object Model are WebLVC:PhysicalEntity and WebLVC:AggregateEntity.

The interaction types defined by Standard WebLVC Object Model are: WebLVC:WeaponFire and WebLVC:MunitionDetonation.

The following sections define the required and optional properties for AttributeUpdate messages and Interaction messages associated with each of the standard object and interaction types.

1. The fact that some intelligent analysis is needed in order to decide on an appropriate JSON representation for each concept is the reason why the WebLVC specification directly defines a Standard Object Model, rather than defining set of automated rules for generating the Standard Object Model from the HLA RPR FOM.

2.2. Standard AttributeUpdate Messages

This section describes the following attribute update messages:

- WebLVC:AggregateEntity
- WebLVC:PhysicalEntity
- WebLVC:EnvironmentalEntity
- WebLVC:RadioTransmitter.

2.2.1. WebLVC:PhysicalEntity Attribute Update Message

The WebLVC:PhysicalEntity attribute message is used to represent the DIS/RPR concept of an individual simulated vehicle or human.

Optional Properties

- EntityIdentifier. DIS-style simulation object identifier expressed as an array of three numbers representing SiteID, ApplicationID, and EntityNumber, for example, [1,2,3].
- EntityType. DIS-style simulation object type expressed as an array of seven numbers representing EntityKind, Domain, CountryCode, Category, Subcategory, Specific, and Extra, as defined by the SISO Enumerations document, for example, [1,2,225,1,6,0,0]. Required only in the first update sent for a particular simulation object, or upon change.
- DeadReckoningAlgorithm. A number representing the dead-reckoning algorithm to use for the simulation object, as defined by the SISO Enumerations document, for example, 4 for "DRM (RVW)".
- WorldLocation, VelocityVector, AccelerationVector. Arrays of three numbers representing the X, Y, and Z components of world location, velocity, and acceleration, as defined by DIS/RPR, for example, [4437182.0232, -395338.0731, 873923.4663].
- Orientation. An array of three numbers representing the psi, theta, and phi components of the simulation object's orientation as defined by DIS/RPR, for example, [-1.65, 2.234, -0.771].
- AngularVelocity. An array of three numbers representing the X, Y, and Z components of angular velocity, as defined by DIS/RPR, for example, [0.37, 1.30, -1.43].
- ForceIdentifier. A number representing the ForceIdentifier of the simulation object, as defined by DIS/RPR.
- Marking. A string representing the marking text of the simulation object, as defined by DIS/RPR.

Appearance Attributes

The RPR FOM defines a large number of appearance attributes as attributes of the PhysicalEntity class or its subclasses. The appearance attributes are defined in the RPR FOM either as Booleans, or as having various enumeration types. For example, DamageState is defined as an enumeration, and IsConcealed is defined as a Boolean. Rather than repeat them all here, this specification includes them by reference. Any of these appearance attributes can be included in a WebLVC entity AttributeUpdate message by including a property with the same name as the corresponding RPR FOM attribute. Use numbers to represent the desired values of attributes that are defined as enumerations, as defined by the SISO Enumerations documents. Use the Boolean values true or false to represent the desired values of attributes that are defined in the RPR FOM as Booleans.

Example Message

The following example WebLVC message defines an attribute update for an object named “F-16 Alpha”, which is of type WebLVC:PhysicalEntity.

```
{
  MessageKind : 1,
  ObjectName : “F-16 Alpha”,
  ObjectType : “WebLVC:PhysicalEntity”,
  EntityIdentifier : [1,2,1],
  EntityType : [1,2,225,1,3,0,0],
  WorldLocation : [4437182.0232, -395338.0731, 873923.4663]
  VelocityVector : [57.04, 32.77, 89.263],
  Orientation : [-1.65, 2.234, -0.771],
  Marking : “F-16”,
  DamageState : 1,
  EngineSmokeOn : true,
  IsConcealed : false
}
```

Notice that this message contains only a subset of the possible properties of physical entities, and that the order of the properties does not matter.

2.2.2. WebLVC:AggregateEntity Attribute Update Message

The WebLVC:AggregateEntity attribute message is used to represent the DIS/RPR concept of a hierarchical unit.

Optional Properties

- EntityIdentifier. DIS-style simulation object identifierexpressed as an array of three numbers representing SiteID, ApplicationID, and EntityNumber, for example, [1,2,3].
- EntityType. DIS-style simulation object type expressed as an array of seven numbers representing EntityKind, Domain, CountryCode, Category, Subcategory, Specific, and Extra, as defined by the SISO Enumerations document, for example, [1,2,225,1,6,0,0]. Required only in the first update sent for a particular unit, or upon change.
- DeadReckoningAlgorithm. A number representing the dead-reckoning algorithm to use for the simulation object, as defined by the SISO Enumerations document, for example, 4 for "DRM (RVW)".
- WorldLocation. An array of three numbers representing the X, Y, and Z components of the center of mass of the unit, as defined by DIS/RPR, for example, [4437182.0232, -395338.0731, 873923.4663].
- VelocityVector. An array of three numbers representing the X, Y, and Z components of world velocity, as defined by DIS/RPR, for example, [89.0232, -38.5, 42.322]
- Orientation. An array of three numbers representing the psi, theta, and phi components of the simulation object's orientation as defined by DIS/RPR, for example, [-1.65, 2.234, -0.771]. Orientation is computed as the average orientation of its subordinates.
- AngularVelocity. An array of three numbers representing the X, Y, and Z components of angular velocity, as defined by DIS/RPR, for example, [0.37, 1.30, -1.43].
- ForceIdentifier. A number representing the ForceIdentifier of the simulation object, as defined by DIS/RPR.
- Marking. A string representing the marking text of the simulation object, as defined by DIS/RPR.
- AggregateState. A number representing the aggregation/disaggregation state of the unit, as defined by DIS/RPR.
- Formation. A number representing the formation of the unit, as defined by DIS/RPR.
- Dimensions. An array of three numbers representing the bounding box that the unit occupies, as defined by DIS/RPR. The values are measured from the center of mass of the unit to the edge of the X, Y, and Z axes of the body coordinate system of the unit, where the axes are aligned with the unit's orientation, for example, [2.0, 2.5, 1.75].

- Subordinates. An array of ObjectName values for those constituent entities that are also represented by individual object instances.
- AggregateSubordinates. List of ObjectName values for those constituent sub-units that are also represented by individual object instances.
- SilentEntities. An array of objects of the form {ObjectType, count} that represent the type and number of subordinate entities that are not represented by individual object instances.
- SilentAggregates. An array of objects of the form {ObjectType, count} that represent the type and number of subordinate units that are not represented by individual object instances.
- SilentEntitiesDamageState. An array of objects of form {ObjectType, DamageState, count} representing the damage status of unpublished subordinate simulation object types. DamageState is the DIS/RPR enumerated value for damage state. Only entries with a damage state other than the default DamageStateNone need to be specified.

Example Message

The following example defines an attribute update for a unit object named “Platoon 1”, which is of type WebLVC:AggregateEntity.

```
{
  MessageKind : 1,
  ObjectName : “Platoon 1”,
  ObjectType : “WebLVC:AggregateEntity”,
  EntityIdentifier : [1,2,3],
  EntityType : [1,1,225,3,2,0,0],
  WorldLocation : [4437182.0232, -395338.0731, 873923.4663]
  Orientation : [-1.65, 2.234, -0.771],
  Marking : “Platoon 1”,
  Dimensions : [10.0, 20.0, 1.0],
  Subordinates : [“Tank1”, “Tank2”, “Tank3”],
  Formation : 3
}
```

2.2.3. WebLVC:EnvironmentalEntity Attribute Update Message

The WebLVC:EnvironmentalEntity attribute message is used to represent the DIS/RPR concept of an environmental process object.

Initial Properties

Type. DIS-style entity type expressed as an array of seven numbers representing EntityKind, Domain, CountryCode, Category, Subcategory, Specific, and Extra, as defined by the SISO Enumerations document, for example, [1,4,0,0,0,0,0]. Required only in the first update sent for a particular environmental entity, or upon change.

Optional Properties

- ProcessIdentifier. DIS-style identifier expressed as an array of three numbers representing SiteID, ApplicationID, and EntityNumber, for example, [1,2,3].
- ModelType. An unsigned integer identifier for the model used for generating this environmental process. The value is exercise-specific.
- EnvironmentProcessActive. Boolean status of the environment process, indicating whether or not it is active.
- SequenceNumber. An integer value used in tagging a series of attribute update messages with unique values. If used, the value should start with zero and increment by one for each update sent. If not used, the value should be set to EP_NO_SEQUENCE.
- GeometryRecords. An array of one or more GeometryRecords, used to describe the physical extent of the object. A GeometryRecord is a JSON object containing the following fields:
  - Type. A string identifying the type of geometry record, `"LineString"`, `"Polygon"`, `"Point"`, or `"Line"`.
  - coordinates. The coordinates of the vertex or vertices of the object. The format of the contents of this field depends on the type and the coordinate reference system. Each coordinate vertex is expressed as an array of three numbers, either [x, y, z] for geocentric Cartesian coordinate reference systems, or [latitude, longitude, altitude] for geodetic coordinate reference systems.

    Case 1: `"Type" = "LineString"`

    coordinates. An array of vertices representing multi-segmented line in world coordinates. Each coordinate is expressed as an array of three numbers.

    Case 2: `"Type" = "Polygon"`

    coordinates. An array of vertices that describe a polygon in world coordinates. Each coordinate is expressed as an array of three numbers.

Case 3: `"Type" = "Point"`

coordinates. A location in world coordinates, expressed as an array of three numbers.

Case 4: `"Type"= "Line"`

coordinates. An array of 2 vertices representing a line segment in world coordinates. Each coordinate is expressed as an array of three numbers.

Example Message

The following example defines an attribute update for an environmental process object with the type `LineString`:

```
{
  "ProcessIdentifier" : [1, 2, 3],
  "Type" : [1, 4, 0, 0, 0, 0, 0],
  "ModelType" : 1,
  "EnvironmentProcessActive" : true,
  "SequenceNumber" : 5,
  "GeometryRecords" : [
    "Type" : "LineString",
    "coordinates" : [
      [4437182.0232, -395338.0731, 873923.4663],
      [4437124.1523, -395341.2841, 873922.5517]
    ]
  ]
}
```

2.2.4. WebLVC:RadioTransmitter Attribute Update Message

The WebLVC:RadioTransmitter attribute update message is used to represent a device that transmits electromagnetic energy in the radio frequency range.

Optional Properties

- HostObjectName. String representing the object that this radio is attached to. This identifier is unique across the simulation.
- EntityIdentifier. DIS-style entity identifier expressed as an array of three numbers representing SiteID, ApplicationID, and EntityNumber, for example, [1,2,3]. Identifies the ID of the radio. This is the same as the host object name when the radio is coming from DIS.
- RadioIndex. A number used to identify the radio transmitter within the scope of the entity.
- RadioEntityType. DIS-style radio type expressed as an array of six numbers representing entity kind domain, country, category, nomenclature version, and nomenclature. See Radio in the SISO Enumerations document.
- TransmitState. A number indicating the state of the transmitter (whether it is off, on but not transmitting, or on and transmitting), as defined by the SISO enumerations document (see Transmit State).
- InputSource. A number representing the source of the radio transmission (for example, pilot, copilot, driver, and so on), as defined in the SISO Enumerations document (see Input Source).
- WorldAntennaLocation. An array of three numbers representing the world location of the radiating portion of the transmitter, as defined by the coordinate reference system, for example, [4437182.0232, - 395338.0731, 873923.4663] for ECEF as in DIS/RPR.
- RelativeAntennaLocation. Arrays of three numbers representing the X, Y, and Z components of relative location of the radiating portion of the transmitter, as defined by DIS/RPR, for example, [1.0, 0.0, -3.0].
- AntennaPatternType. A number representing the radiation pattern of the antenna, as defined in the SISO Enumerations document (see Antenna Pattern Type). Its value determines the content of the AntennaPatternParameters property.
- Frequency. A number representing the center frequency used by the radio in transmission. Expressed as a positive integer number in units of hertz.
- TransmitBandwidth. A number representing the bandpass of the radio, expressed as a position floating point number in units of hertz.
- Power. A number representing the average power being transmitted in units of decibel-milliwatts.

- ModulationType. A JSON object representing the type of modulation used for radio transmission. The properties of this object are:
  - SpreadSpectrum. A number representing the spread spectrum technique in use, as defined in the SISO Enumerations document.
  - Major. A number representing the major classification of the modulation type, as defined in the SISO Enumerations document.
  - Detail. A number representing certain detailed information depending on the major modulation type, as defined in the SISO Enumerations document.
  - System. A number used to specify the interpretation of the Modulation Parameter fields in the RadioTransmitterUpdate message, as defined in the SISO Enumerations document.
- CryptoMode. A number representing the crypto mode (baseband or phase), as defined in the SISO Enumerations document.
- CryptoSystem. A number representing the use of crypto or secure voice equipment, as defined in the SISO Enumerations document.
- CryptoKey. A number representing the encryption key (if encryption is in use), zero otherwise.
- AntennaPatternParameters. A JSON object that may be present, depending on the value of the AntennaPatternType property (DIS/RPR Beam Antenna Pattern record):
  - BeamDirection. An array of three numbers representing the psi, theta, and phi components of the beam direction as defined by DIS/RPR, e.g. [-1.65, 2.234, -0.771].
  - AzimuthBeamwidth. A number representing the full width of the beam to the -3 dB power density points in the x-y plane of the beam coordinate system, in radians.
  - ElevationBeamwidth. A number representing the full width of the beam to the -3 dB power density points in the x-z plane of the beam coordinate system, in radians.
  - ReferenceSystem. A number specifying the reference coordinate system with respect to which the beam direction is specified, as defined in the SISO Enumerations document.
  - Ez. A number representing the DIS/RPR concept of the magnitude of the Z-component (in beam coordinates) of the Electrical field.
  - Ex. A number representing the DIS/RPR concept of the magnitude of the X-component (in beam coordinates) of the Electrical field.
  - Phase. A number representing the phase angle between Ez and Ex in radians.
- FrequencyHopInUse. A Boolean indicating whether or not frequency hopping is in use.
- PseudoNoiseInUse. A Boolean indicating whether or not pseudo noise is in use.
- TimeHopInUse. A Boolean indicating whether or not time hopping is in use.

Example Message

```
{
  “MessageKind” : 1,
  “EntityIdentifier” : [1, 1, 3001],
  “RadioIndex” : 1,
  “RadioEntityType” : [225, 2, 3, 4],
  “TransmitState” : 1,
  “RadioEntityType” : [1, 1, 225, 3, 4, 5],
  “InputSource” : 2,
  “WorldAntennaLocation” : [4437182.0232, -395338.0731,873923.4663],
  “RelativeAntennaLocation” : [1.0, 0.0, -3.0],
  “AntennaPatternType” : 1,
  “Frequency” : 44056,
  “Power” : 50.5,
  “ModulationType” : {
    “SpreadSpectrum” : 1,
    “Major”: 2,
    “Detail” : 3,
    “System” : 4
  },
  “CryptoMode” : 1,
  “CryptoSystem” : 4,
  “CryptoKey” : 2348238752,
  “AntennaPatternParameters” : {
    “BeamDirection” : [-1.65, 2.234, -0.771],
    “AzimuthBeamwidth” : 0.25,
    “ElevationBeamwidth” : 0.78,
    “ReferenceSystem” : 2,
    “Ez” : 2.533,
    “Ex” : 1.29,
    “Phase” : 0.707
  },
  “FrequencyHopInUse” : true,
  “PseudoNoiseInUse” : false,
  “TimeHopInUse” : true
}
```

2.3. Standard Interaction Messages

This section describes the following interaction messages:

- WebLVC:WeaponFire
- WebLVC:MunitionDetonation
- WebLVC:StartResume
- WebLVC:StopFreeze
- WebLVC:RadioSignal.

2.3.1. WebLVC:WeaponFire Interaction Message

The WebLVC:MunitionDetonation message represents the firing of a weapon.

Optional Properties

- AttackerId. The ObjectName of the object that fired the weapon.
- TargetId. The ObjectName of the object that was targeted by the attacker, if applicable.
- MunitionType. The type of the munition object. A DIS-style simulation object type expressed as an array of seven numbers representing EntityKind, Domain, CountryCode, Category, Subcategory, Specific, and Extra, as defined by the SISO Enumerations document, for example, [1,2,225,1,6,0,0].
- MunitionId. The ObjectName of the munition, if it is simulated as an entity.
- EventId. A string identifier generated by the issuing federate, used to associate related fire and detonation events.
- WarheadType. A number representing the type of warhead fitted to the munition being fired, as defined by DIS/RPR.
- FireMissionIndex. A unique number to identify the fire mission. Used to associate weapon fire interactions in a single fire mission.
- FuseType. A number representing the type of fuse on the munition, as defined by DIS/RPR. This field may be omitted if undefined.
- Quantity. The number of rounds fired in the fire event. A value of zero indicates a single round.
- Rate. A number representing the rate at which munitions are discharged in rounds per minute.
- Range. A number, representing the range in meters assumed by the firing simulation object in computing the fire control solution.
- Location. The location in geocentric coordinates from which the munition was launched, expressed as an array of three numbers, for example, [4437182.0232, -395338.0731, 873923.4663].

- Velocity. The velocity of the munition in geocentric coordinates, at the point of the muzzle flash, represented as an array of three numbers, for example, [89.0232, -38.5, 42.322].

Example Message

The following example defines an interaction of WebLVC:WeaponFire.

```
{
  MessageKind : 2,
  InteractionType : "WebLVC:WeaponFire",
  AttackerId : "Tank1",
  TargetId : "Tank2",
  MunitionType : [2,2,225,2,3,0,0],
  Location : [4437182.0232, -395338.0731, 873923.4663],
  Velocity : [57.04, 32.77, 89.263]
}
```

2.3.2. WebLVC:MunitionDetonation Interaction Message

The WebLVC:MunitionDetonation message represents the detonation of a munition.

Optional Properties

- AttackerId. The ObjectName of the object that fired the weapon.
- TargetId. The ObjectName of the object that was targeted by the attacker. For indirect fire this field may be omitted.
- MunitionId. The ObjectName of the munition, if it is simulated as an entity object. If not, then this field may be omitted.
- MunitionType. The type of the munition object. A DIS-style simulation object type expressed as an array of seven numbers representing EntityKind, Domain, CountryCode, Category, Subcategory, Specific, and Extra, as defined by the SISO Enumerations document, for example, [1,2,225,1,6,0,0].
- EventId. A string identifier generated by the issuing federate, used to associate related fire and detonation events.
- Velocity. The velocity of the munition in geocentric coordinates, at the time of detonation, expressed as an array of three numbers, for example, [89.0232, -38.5, 42.322].
- WorldLocation. Location of the detonation with respect to the world in geocentric coordinates, expressed as an array of three numbers, for example, [4437182.0232, -395338.0731, 873923.4663].
- EntityLocation. Location of the detonation with respect to the target simulation object's coordinate system, expressed as an array of three numbers, for example, [-1.4, 2.7, 0.6].
- Result. A number representing the result of the detonation as specified by DIS/RPR.

- FuseType. A number representing the type of fuse on the munition, as defined by DIS/RPR.
- Quantity. The number of rounds fired in the fire event. A value of zero indicates a single round.
- Rate. A number representing the rate at which munitions are discharged in rounds per minute.
- WarheadType. A number representing the type of warhead fitted to the munition being fired, as defined by DIS/RPR.

Example Message

The following example defines an interaction of WebLVC:MunitionDetonation.

```
{
  MessageKind : 2,
  InteractionType : “WebLVC:MunitionDetonation”,
  AttackerId : “Tank1”,
  TargetId : “Tank2”,
  MunitionType : [2,2,225,2,3,0,0],
  WorldLocation : [4437182.0232, -395338.0731, 873923.4663],
  Result : 1
}
```

2.3.3. WebLVC:StartResume Interaction Message

The WebLVC:StartResume message directs one or more simulators to start or resume simulating, or start or resume simulation of a specific entity.

Required Properties

- ReceivingEntity. The DIS/RPR simulation ID of the receiving simulations, or the entity ID of a specific entity, expressed as an array of three numbers. The allowable values for the ID follow the DIS/RPR rules.
- RequestIdentifier. An integer used to identify a particular request. The value should be incremented for each request.
- RealWorldTime. A number representing the real world time of the request, as defined by DIS/RPR.
- SimulationTime. A number representing the time of day in simulation time of the request, as defined by DIS/RPR.

Optional Properties

OriginatingEntity. The DIS/RPR simulation ID (Site ID, Application ID) of the simulator requesting the start/resume action, expressed as an array of two numbers.

2.3.4. WebLVC:StopFreeze Interaction Message

The WebLVC:StopFreeze message directs one or more simulators to stop simulating, or stop simulation of a specific entity.

Required Properties

ReceivingEntity. The DIS/RPR simulation ID of the receiving simulation(s), or the entity ID of a specific entity, expressed as an array of three numbers. The allowable values for the ID follow the DIS/RPR rules.

Optional Properties

- OriginatingEntity. The DIS/RPR simulation ID (Site ID, Application ID) of the simulator requesting the stop/freeze action, expressed as an array of two numbers.
- RealWorldTime. A number representing the real world time of the request, as defined by DIS/RPR.
- Reason. A number indicating the reason for stopping the simulation. Allowed values are an enumeration defined by DIS/RPR.
- ReflectValues. A Boolean indicating whether the entity or entities should continue to reflect attributes while frozen.
- RunInternalSimulationClock. A Boolean indicating whether the entity or entities should continue to run their internal simulation clocks while frozen.
- UpdateAttributes. A Boolean indicating whether the entity or entities should continue to update attributes while frozen.

2.3.5. WebLVC:RadioSignal Interaction Message

The WebLVC:RadioSignal interaction message represents the wireless transmission and reception of audio or digital data via electromagnetic waves.

Optional Properties

- RadioIdentifier. A string representing the particular transmitter that is transmitting. This identifier is unique across the simulation.
- EncodingClass. A number representing the encoding scheme being used, as defined by the SISO Enumerations document (see the Radio signal encoding class).
- EncodingType. A number representing the type of TDL message. If set to zero, this is not a TDL message. When this is a positive number, the number represents the type of TDL message, as defined by the SISO Enumerations document scheme being used, as defined by the SISO Enumerations document as (see the Radio signal encoding type).
- TDLType. A number representing the type of TDL message included in the signal message, as defined by the SISO Enumerations document (see TDL Type).
- SampleRate. A number representing sample rate in samples per second for audio data. The data rate is in bits per second for digital data.
- SampleCount. A number representing the number of samples in the audio data.
- SampleData. A base64-encoded string representing the data content of the message.
- DatabaseIndex. A number representing the index of a prerecorded audio file that is to be looked up in a known database for playing.
- UserProtocolID. The protocol id of the data stream.

Example:

```
{
  “MessageKind” : 2,
  “InteractionType” : “WebLVC:RadioSignalInteraction”,
  “RadioIdentifier” : ““[1, 1, 3001]-1”,
  “EncodingClass” : “Tank2”,
  “EncodingType” : 0,
  “SampleRate” : 44056,
  “SampleData” : “base64:...”
}
```

3. MÄK Extensions to the Standard WebLVC Object Model

MÄK has extended the standard WebLVC object model with generic messages and application-specific messages.

3.1. Introduction

MÄK has extended the Standard WebLVC Object Model to support communication with specific simulation applications. The data exchanged by these applications is not general enough in some cases to be incorporated in the standard object model. Therefore, MÄK has defined a set of extensions to the Standard WebLVC Object Model to facilitate control of specific applications, including MÄK Logger, VR-Forces, and VR-Vantage.

To distinguish between the MÄK-specific messages and the standard WebLVC object model messages, by convention all object and interaction type strings are prefaced by the string MAK: instead of WebLVC:.

3.2. The MÄK Data Logger Control Message

The MAK:LoggerControl interaction message is used to control the MÄK Data Logger application.

Required Properties

- Target. The DIS/RPR simulation ID (Site ID, Application ID, Remote Control ID) of the MÄK Logger application to be controlled, expressed as an entity identifier string.
- Command. The Logger control command to execute, expressed as a string. Table 3-1 describes the commands that a Logger message can send and provides an example for each one.

Table 3-1: Logger commands (Sheet 1 of 3)

Name	Description
loggerPlayback	Play the current Logger file. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCPlay", "Target" : "1:1:1" }`
loggerPause	Pause the current Logger action. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCPause", "Target" : "1:1:1" }`
loggerStart	Start recording. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCStart", "Target" : "1:1:1" }`
loggerPlayFile	Load and play a Logger file. By default the path is logger_tapes. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCPlayFile", "Target" : "1:1:1", "Filename" : "WebLVCFile.lgr" }`
loggerRecordFile	Initialize an empty file and start recording. By default the path is logger_tapes. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCRecordFile", "Target" : "1:1:1", "Filename" : "WebLVCFile.lgr" }`
loggerDelFile	Delete a Logger file. By default the path is logger_tapes. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCDelFile", "Target" : "1:1:1", "Filename" : "WebLVCFile.lgr" }`

Table 3-1: Logger commands (Sheet 2 of 3)

Name	Description
loggerEmpty	Unloads any existing files. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCEmpty", "Target" : "1:1:1" }`
loggerQuit	Shut down the Logger. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCQuit", "Target" : "1:1:1" }`
loggerEnd	Jump to the end of the loaded Logger file. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCEnd", "Target" : "1:1:1" }`
loggerStop	Stop playback or record. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCStop", "Target" : "1:1:1" }`
loggerRecord	Begin recording. A file must be loaded. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCRecord", "Target" : "1:1:1" }`
loggerSetTime	Set the time in seconds of the currently loaded file in the Logger. `{ "MessageKind": 2, "InteractionType": "MAK:LoggerControl", "Command" : "LCSetTime", "Target" : "1:1:1", "Time": 5.0 }`
loggerSkipTime	Skip a number of seconds in the currently loaded file.

Table 3-1: Logger commands (Sheet 3 of 3)

Name	Description

	<pre>{
	"MessageKind": 2,
	"InteractionType": "MAK:LoggerControl",
	"Command" : "LCSkipTime",
	"Target" : "1:1:1",
	"Time": 5.0
	}</pre>

loggerTimeScale	Set the playback speed of the Logger. Time is a speed multiplier, where 1 is real time.
	<pre>{
	"MessageKind": 2,
	"InteractionType": "MAK:LoggerControl",
	"Command" : "LCTimeScale",
	"Target" : "1:1:1",
	"TimeScale": 2.0
	}</pre>

loggerPlayAction	Tells a Logger what action to take initially when set to playback mode. Valid values are '0' for play and '3' for stop.
	<pre>{
	"MessageKind": 2,
	"InteractionType": "MAK:LoggerControl",
	"Command" : "LCPlayAction",
	"Target" : "1:1:1",
	"Action": 0
	}</pre>

loggerRecordAction	Tells a Logger what action to take initially when set to record mode. Valid values are '2' for record and '3' for stop.
	<pre>{
	"MessageKind": 2,
	"InteractionType": "MAK:LoggerControl",
	"Command" : "LCRecordAction",
	"Target" : "1:1:1",
	"Action": 2
	}</pre>

loggerEofAction	Tells a Logger what action to take when a played file reaches the end of the file. Valid values are '5' for loop and '3' for stop.
	<pre>{
	"MessageKind": 2,
	"InteractionType": "MAK:LoggerControl",
	"Command" : "LCEofAction",
	"Target" : "1:1:1",
	"Action": 5
	}</pre>

3.3. MAK:VRFBackend Attribute Update Message

MÄK adds one object type, MAK:VRFBackend. It represents a VR-Forces simulation engine application.

Optional Properties

- ControlState. A number representing the simulation's run/pause state. Allowed values are:
  - 1. Simulation back-end is paused.
  - 2. Simulation back-end is running.
- ExerciseStartTime. A number indicating the simulation start time of the current scenario, in seconds.
- GuiTerrainName. The path to the terrain database file loaded by the VR-Forces graphical user interface (GUI). The path can be absolute or relative to the location of the VR-Forces simulation engine executable.
- InetAddr. A string representing the IP address of the VR-Forces simulation engine (for example, "192.168.1.101").
- IsCurrentParameterDb. Boolean indicating whether or not the VR-Forces object parameters database in use is in the current file format.
- IsLoaded. Boolean indicating whether or not the VR-Forces simulation engine has a scenario loaded.
- IsTimeManaged. Boolean indicating whether or not the VR-Forces simulation engine is running in time-managed mode.
- LoadedNewScenario. Boolean indicating whether a new scenario has just been loaded by the VR-Forces simulation engine. It is set to true upon loading a scenario, and published exactly once for the duration of that scenario being loaded in VR-Forces. Any subsequent updates will have this value set to false.
- ScenarioName. The path to the VR-Forces scenario file. The path can be absolute or relative to the location of the VR-Forces simulation engine executable.
- SimEngineId. The DIS/RPR simulation ID (Site ID, Application ID) of the VR-Forces simulation engine, expressed as an array of two numbers.
- SimStartTime. The simulation start time of the current scenario, in seconds.
- SimTime. The current simulation time in seconds.
- TimeMultiplier. A number used to scale simulation time for faster or slower than real-time simulation.

3.4. VR-Forces Interaction Messages

MÄK has added the following messages for controlling VR-Forces:

- MAK:VrfControl
- MAK:VrfCurrentResourceRequest
- MAK:VrfResourceMonitorResponse
- MAK:VrfScriptedTask
- MAK:VrfSet
- MAK:VrfTextReport
- MAK:VrfChangeTimeOfDay
- MAK:VrfChangeGlobalWeather
- MAK:VrfCreateObject
- MAK:VrfScenario.

VR-Forces uses the following terminology to refer to the objects it simulates:

- Entity. An individual object such as a truck, airplane, or human.
- Unit. A group of individual entities or other units treated as one organizational unit, such as a platoon, a company, or a crowd.
- Simulation object. Generically refers to both entities and units.

3.4.1. MAK:VrfCurrentResourceRequest Interaction Message

The MAK:VrfCurrentResourceRequest interaction message is used to request information about a VR-Forces simulation object's resources (for example, fuel, munitions, and so on). A MAK:VrfResourceMonitorResponse message is sent in response by a VR-Forces simulation engine, containing the set of resources owned by the simulation object.

Required Properties

- OriginatingEntity. The DIS/RPR simulation ID (Site ID, Application ID) of the application requesting the resource information, expressed as an array of two numbers.
- ReceivingEntity. The DIS/RPR simulation ID (Site ID, Application ID) of the VR-Forces simulation engine responsible for simulating the simulation object for which the caller is requesting resource information, expressed as an array of two numbers.
- ObjectName. The ObjectName of the simulation object for which the caller is requesting resource information.
- RequestIdentifier. An integer used to identify a particular request. The value should be incremented for each request. The value of the RequestIdentifier will be included in the MAK:VrfResourceMonitorResponse message.

3.4.2. MAK:VrfResourceMonitorResponse Interaction Message

The MAK:VrfResourceMonitorResponse interaction message contains information about the resources maintained by a VR-Forces simulation object (for example, fuel, munitions, and so on). The VR-Forces simulation engine responsible for simulating the simulation object sends this message in response to a MAK:VrfCurrentResourceRequest message.

Required Properties

- ObjectName. The VR-Forces ObjectName of the simulation object for which the caller is requesting resource information.
- Resources. An array of objects containing information about the simulation object's resources. Each object contains information about a specific resource. The array may be empty, depending on whether or not resources are modeled for the simulation object. Consult the VR-Forces documentation for details about how resources are defined. The object is structured as follows:
  - ResourceName. A string name that identifies the resource, for example, "fuel".
  - CurrentAmount. A number representing the amount of the resource available to the simulation object. The unit of measure depends on the type of resource, as defined in the VR-Forces simulation engine. For example, the amount of a fuel resource is typically expressed in terms of liters, while the amount of an ammunition resource would typically be the number of rounds.
  - FullAmount. A number representing the maximum capacity for this resource on the simulation object. Its unit of measure is the same as that of CurrentAmount.
  - ResourceType. A string describing the numeric type of resource. ResourceType indicates whether or not the CurrentAmount and FullAmount properties represent integer or floating-point numbers. Values of `integer-resource` and `munition-resource` indicate that the resource amounts are expressed in terms of integers. A value of `munition-resource` is specified for an integer resource when the resource represents some type of munition. A value of `real-resource` indicates that resource amounts are expressed in terms of double-precision floating-point numbers.
  - ResourceEntityType. DIS-style simulation object type of the resource, expressed as an array of seven numbers representing Entity Kind, Domain, Country Code, Category, Subcategory, Specific, and Extra, as defined by the SISO Enumerations document, for example, [1,4,0,0,0,0,0]. Only those resources that are published as separate entities (such as missiles) specify this property.

3.4.3. MAK:VrfScriptedTask Interaction Message

The MAK:VrfScriptedTask message is used to direct a VR-Forces simulation object to perform a task, such as move-to-location or fire-weapon. The message has a flexible format and can represent a wide range of tasks provided by VR-Forces.

Required Properties

- ReceivingApplicationName. A string containing the object name of the simulation object to be tasked.
- Subtask. A Boolean indicating whether or not the task should be executed as a subtask. This flag is set to True when the task to be executed is to be done in support of a higher-level task, and is typically issued by a VR-Forces simulation object to itself. An external application sending a MAK:VrfScriptedTask should always set this value to False.
- ScriptedTask. A JSON object that contains the data for the task. The object contains the following properties:
  - ScriptID. A string that identifies the type of VR-Forces task. See "Scripted Task Documentation" in *VR-Forces Developer's Guide*, for details on the types of tasks available. The following ScriptID values are supported, as of VR-Forces 4.4:
    - AircraftHold
    - AircraftHold_Waypoint
    - animated-movement-task
    - Arm_Mine_At_Depth
    - Avoid_Other_Ships
    - Bombing_Mission
    - Bounding_Overwatch_Move_To
    - Create_and_Move_To_Waypoint
    - Delayed_homing
    - Delayed_homing_ASW
    - Deploy_Sonobuoy
    - Deploy_Sonobuoys_Along_Route
    - Drop_Naval_Depth_Charge
    - Drop_Naval_Depth_Charge_At_Location
    - Drop_Naval_Mine
    - Drop_Naval_Mines_Along_Route
    - Evade_Missile
    - Explode_Charge_At_Depth
    - Fixed_Wing_Land
    - Fixed_Wing_Takeoff

- goto-depth
- launch-munition
- Launch_ASW_Missile
- Lower_Periscope
- Make_Way_for_Emergency
- Monitor_Terrain
- Move In Cover
- move-on-roads-to-location
- move-on-roads-to-waypoint
- move-to-location-path-plan
- move-to-waypoint-path-plan
- Move_to_Named_Street
- Naval_Mine_Sweep
- navigate-to-location
- navigate-to-waypoint
- Orbit__script_
- Place_IED
- Raise_Periscope
- SAR_Creep_Pattern
- SAR_Parallel_Pattern
- SAR_Sector_Search
- ship-navigate-to-location
- ship-navigate-to-waypoint
- Sonar_Dip
- stop-moving-ship
- Talk_To_Friends
- turn-move-sail-heading
- turn-move
- ZigHeading

– ScriptVariables. An array of zero or more objects representing the parameters associated with a given task type (identified by the ScriptID). The content of this array varies according to the type of task. For information about the parameters expected for a given task type, see Section 12.5 - Starting and Processing C++ and Scripted Tasks, in *VR-Forces Developer's Guide*.

An object in the ScriptVariables array represents a task parameter and is comprised of the following properties:

- name. A string that identifies the name of the task parameter.
- type. A string that identifies the type of data for the task parameter. See Section 12.5 - Starting and Processing C++ and Scripted Tasks, in *VR-Forces Developer's Guide*, for details about the types of tasks available.
- value. A string, number, object, Boolean, or array that holds the value of the task parameter. The kind of JSON object used in this property depends on the value of the type property. Table 3-2 lists types and JSON property types.

Table 3-2: Types referenced to JSON property types

Type	Type of JSON Object	Range	Meaning
Boolean	Boolean	true or false	
ChoiceDropdownList	Number	A positive number starting with zero.	Index into a list of options.
ChoiceOptionButton	Number	A positive number starting with zero.	Index into a list of options.
DIGuyAnimation	Number		
Emitter	Number		
Force	Number		
Integer	Number		
MunitionResource	Number		
Resource	Number		
LocationWithAltitude	Array of three numbers representing world coordinates.		
LocationWithoutAltitude	Array of three numbers representing world coordinates.		
OffsetLocation	Array of three numbers representing world coordinates.		
BombResource	String		
String	String		
SimulationObjectSingle	String		
SimulationObjectMultiple	String		
EntityType	String		

Table 3-2: Types referenced to JSON property types

Type	Type of JSON Object	Range	Meaning
AggregateFormation	String		
AltitudeMSL	Number		
DepthMSL	Number		
Distance	Number		
Heading	Number		
Real	Number		
Speed	Number		
Time	Number		
TurnRate	Number		
Unknown	Undefined		

Optional Properties

OriginatingApplicationName. A string identifying the sending application.

3.4.4. MAK:VrfSet Interaction Message

The MAK:VrfSet message is used to direct a VR-Forces simulation object to set some aspect of its state.

Required Properties

- ReceivingApplicationName. The object name of the VR-Forces simulation object whose state is to be set, or the string DtBroadcast to designate all VR-Forces simulation objects.
- DataVariables. An array of one or more objects. Each object represents an individual request to set state in a VR-Forces simulation object. An object in this array contains the following required property:
  - name. A string identifying the particular aspect of state to be set. The allowed values are:
    - Destroy. Set the damage status of the simulation object to destroyed.
    - Restore. Restore the simulation object to full health and replenish resources.
    - Speed. Set the ordered speed of the simulation object. The simulation object will attempt to maintain this speed in any movement behaviors it executes.
    - ReactiveTaskCancel. Cancel the current reactive task the simulation object is executing (if any). For more information about reactive tasks, please see *VR-Forces Users Guide*.

  An object in this array may contain the following optional properties:

  - type. A string identifying the type of value to set for state. See "Scripted Task Documentation" in *VR-Forces Developer's Guide*, for a list of the possible values for type.
  - value. A string, Boolean, number, object, or array containing the value of the state to set. Table 3-3 lists the expected value/types for each of the given MAK:VrfSet name strings.

Table 3-3: Expected value/types for MAK:VrfSet strings

Name	Type	Value
Destroy	N/A (property not used)	N/A (property not used).
Restore	N/A (property not used)	N/A (property not used).
Speed	Speed	A positive real number in meters/second.
ReactiveTaskCancel	String	The ScriptId of the task to be canceled.

Optional Properties

OriginatingApplicationName. A string identifying the name of the sending application.

3.4.5. MAK:VrfTextReport Interaction Message

The MAK:VrfTextReport message is used to send a VR-Forces DtTextReport message from one simulation object to another simulation object. For details, please refer to *VR-Forces Developer's Guide* section 11 "Entity Communication". An example is given in section 11.2 "Sending Messages".

Required Properties

- message. a string containing the message to be sent.
- originatingEntity. the object name of the VR-Forces simulation object sending the message.
- receivingEntity. the object name of the VR-Forces simulation object to receive the message, or the string DtBroadcast to designate all VR-Forces simulated entities.

3.4.6. MAK:VrfChangeTimeOfDay

The MAK:VrfChangeTimeOfDay message changes the current time of day inside VR-Forces.

Required Properties

- SecondsPastMidnight. Highlights the new time of day.
- TimeOfDayAdvanceSpeed. Set to 1 if the scenario is currently running or 0 if the scenario is paused.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrfChangeTimeOfDay",
  "SecondsPastMidnight": 3600,
  "TimeOfDayAdvanceSpeed": 1.0
}
```

3.4.7. MAK:VrfChangeGlobalWeather

The MAK:VrfChangeGlobalWeather message changes the weather in VR-Forces. For full details on what each setting does, see the DtChangeWeatherCommand documentation in VR-Forces. All of the settings are optional. You can select only a subset of these and only that subset of weather will change.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrfChangeGlobalWeather",
  "AirTurbidity" : 1.0,
  "AmbientAirTemperature" : 1.0,
  "CloudState" : 1,
  "PrecipitationIntensity" : 1.0,
  "PrecipitationType" : 1,
  "Visibility" : 1.0,
  "WindDirection" : 1.0,
  "WindSpeed" : 1.0,
  "UnderwaterVisibility" : 1.0,
  "SurfaceTransparency" : 1.0,
  "WaterCurrentSpeed" : 1.0,
  "WaterCurrentDirection" : 1.0,
  "AmbientWaterTemperature" : 1.0,
  "TidalOffset" : 1.0,
  "TidalDirection" : 1.0,
  "SwellState" : 1,
  "SwellStateDirection" : 1.0,
  "SeaState" : 1,
  "SeaStateDirection" : 1.0,
  "WindDrivenSeaStateEnabled" : true,
  "Choppiness" : 1.0,
  "SurgeDepth" : 1.0,
  "ThermoclineLayer1Depth" : 1.0,
  "ThermoclineLayer1Permeability" : 1.0,
  "ThermoclineLayer2Depth" : 2.0,
  "ThermoclineLayer2Permeability" : 2.0,
  "ThermoclineLayer3Depth" : 3.0,
  "ThermoclineLayer3Permeability" : 3.0,
  "ThermoclineLayer4Depth" : 4.0,
  "ThermoclineLayer4Permeability" : 4.0,
  "ThermoclineLayer5Depth" : 5.0,
  "ThermoclineLayer5Permeability" : 5.0
}
```

3.4.8. MAK:VrfCreateObject

The MAK:VrfCreateObject message creates a VR-Forces object.

Required Properties

- Address. A DtSimulationAddress identifying the particular VR-Forces back-end to create the object.
- Name. A string used to uniquely identify the object. This string must be unique for all objects in the exercise.
- ObjectType. The type of the new simulation object. This is a 7-tuple number that matches the DIS/RPR Entity Type value.
- ForceType. The force value for the new simulation object. 0 is other, 1 is friendly, 2 is opposing, 3 is neutral.
- Label. A string associated with the new object. This string provides an additional means to identify an object without the strict uniqueness requirement of object name.
- Vertices. A list of the vectors that make up the object's geometry (for example, the vertices that make up the points along a route). For simulation object position, the position is just an alias for the first vertex. Vertices must be specified in geocentric coordinates.

Optional Properties

- Heading. A double value that is the initial heading of the object in radians. A value of 0.0 is north and positive values are clockwise. Default: 0.0.
- Projection. A bool flag indicating whether or not the Z elements in the object's vertex list have meaning. For example, phase lines do not have meaningful Z elements. Default: false.
- ClosedFigure. A bool flag indicating whether the object's vertices make up a closed figure (that is, does the last vertex connect with the first). A control area is an example of an object that is a closed figure. Default: false.
- CreateSubObjects. A bool flag indicating whether or not you want to create a sub-object. The parameter file may contain a list of subobjects. When building from the bottom up (for example, using AggregateAs or explicitly calling addSubordinate) then the subobjects in the parameter list should be ignored. Default: false.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrfCreateObject",
  "Address" : [1,3001],
  "Name" : "M1A2 1",
  "ObjectType" : [1,1,225,1,1,3,0],
  "ForceType" : 1,
  "Label" : "My Tank",
  "Heading" : 0.0,
  "Vertices" : [[1526692.0817458, -4464873.1812864, 4276853.4378026]],
  "Projection" : false,
  "ClosedFigure" : false,
  "CreateSubObjects" : false
}
```

3.4.9. MAK:VrfScenario

The MAK:VrfScenario message changes the current scenario in VR-Forces.

Required Properties

Command. The action to take. Options are `new`, `load`, `save`, and `close`.

Optional Properties

ScenarioName. Specifies a scenario. If a scenario name is specified, the server tries to find that scenario in the current path. If it fails, it looks for an environment variable MAK_VRFScenarioDir or MAK_VRFTerrainDir (for new calls). If that variable does not exist, it looks for the default installation location of VR-Forces.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrfScenario",
  "Command" : "load",
  "ScenarioName" : "RaidDemo/Raid Demo.scn"
}
```

3.4.10. MAK:VrfControl

The MAK:VrfControl message controls the current scenario state of VR-Forces. A scenario must be loaded for this to work.

Required Properties

ControlType. Possible ControlTypes are `Pause`, `Run`, `Rewind`, `Step`, `RunDuration`, and `RunComplete`.

Optional Properties

Time. RunDuration and RunComplete require a Time value. Otherwise, it is not used.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType" : "MAK:VrfControl",
  "ControlType" : "Rewind",
  "Time": 0.0
}
```

3.5. VR-Vantage Messages

MÄK has added the following messages for controlling VR-Vantage:

- MAK:VrvMimicView
- MAK:VrvMimicTrackView
- MAK:VrvSensorMode
- MAK:VrvViewMagnification
- MAK:VrvObserverMode
- MAK:VrvUseSavedView
- MAK:VrvSetSystemState
- MAK:VrvModelSet
- MAK:VrvModelScaling
- MAK:VrvModelScaleFactor
- MAK:VrvKeyPress
- MAK:VrvFollowView
- MAK:VrvTetherView
- MAK:VrvMouseEvent
- MAK:VrvTrackHistories
- MAK:VrvDynamicOcean.

The VR-Vantage messages can also control a VR-Forces front-end.

3.5.1. MAK:VrvMimicView

The MAK:VrvMimicView message attaches an observer to a simulation object using Mimic mode.

Required Properties

AttachedEntity. The simulation object to attach to.

Optional Properties

Observer. Specifies the active observer.

Offset. Specifies an offset from the location of the simulation object you are attaching to. It is specified in body-centric coordinates with Z up, Y to the front, and X to the right.

OffsetOri. Specifies an offset from the location of the simulation object you are attaching to. It is a body-centric taitBryan where psi rotates to the left, theta looks up, and phi rolls to the right.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvMimicView",
  "Observer": "Observer 1",
  "AttachedEntity": "M1A2 1",
  "Offset": [ 2.0, -10.0, 0.0 ],
  "OffsetOri": [ 0.0, 0.0, 0.0 ]
}
```

3.5.2. MAK:VrvMimicTrackView

The MAK:VrvMimicTrackView message attaches an observer to a simulation object using Mimic Track mode.

Required Properties

AttachedEntity. The simulation object to attach to.

TrackedEntity. The simulation object that is tracked.

Optional Properties

Observer. Specifies the active observer.

Offset. Specifies an offset from the location of the simulation object you are attaching to. It is specified in body-centric coordinates with Z up, Y to the front, and X to the right.

OffsetOri. Specifies an offset from the location of the simulation object you are attaching to. It is a body-centric taitBryan where psi rotates to the left, theta looks up, and phi rolls to the right.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvMimicTrackView",
  "Observer": "Observer 1",
  "AttachedEntity": "M1A2 1",
  "TrackedEntity": "M1A2 2",
  "Offset": [ 2.0, -10.0, 0.0 ],
  "OffsetOri": [ 0.0, 0.0, 0.0 ]
}
```

3.5.3. MAK:VrvFollowView

The MAK:VrvFollowView message attaches an observer to an simulation object using Follow mode.

Required Properties

AttachedEntity. The simulation object to attach to.

Optional Properties

Observer. Specifies the active observer.

Offset. Specifies an offset from the location of the simulation object you are attaching to. It is specified in body-centric coordinates with Z up, Y to the front, and X to the right.

OffsetOri. Specifies an offset from the location of the simulation object you are attaching to. It is a body-centric taitBryan where psi rotates to the left, theta looks up, and phi rolls to the right.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvFollowView",
  "Observer": "Observer 1",
  "AttachedEntity": "M1A2 1",
  "Offset": [ 2.0, -10.0, 0.0 ],
  "OffsetOri": [ 0.0, 0.0, 0.0 ]
}
```

3.5.4. MAK:VrvTetherView

The MAK:VrvTetherView message attaches an observer to an simulation object using Tether mode.

Required Properties

AttachedEntity. The simulation object to attach to.

Optional Properties

Observer. Specifies the active observer.

Offset. Specifies an offset from the location of the simulation object you are attaching to. It is specified in body-centric coordinates with Z up, Y to the front, and X to the right.

OffsetOri. Specifies an offset from the location of the simulation object you are attaching to. It is a body-centric taitBryan where psi rotates to the left, theta looks up, and phi rolls to the right.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvTetherView",
  "Observer": "Observer 1",
  "AttachedEntity": "M1A2 1",
  "Offset": [ 2.0, -10.0, 0.0 ],
  "OffsetOri": [ 0.0, 0.0, 0.0 ]
}
```

3.5.5. MAK:VrvSensorMode

The MAK:VrvSensorMode message switches the sensor mode for an observer and a specific version of VR-Vantage.

Required Properties

Vantage. Specifies the instance of VR-Vantage by the Site, App, and Exercise ID triple.

SensorMode. The mode to change to. SensorMode options are `Sensor Off`, `NVG`, `IR`, and `IR (Black Hot)`.

Optional Properties

Observer. Specifies the active observer.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvSensorMode",
  "Observer": "Observer 1",
  "Vantage": [ 2, 1, 1 ],
  "SensorMode": "NVG"
}
```

3.5.6. MAK:VrvObserverMode

The MAK:VrvObserverMode message switches the observer mode for an observer.

Required Properties

ObserverMode. Specifies the observer mode: `Stealth`, `XR`, `Plan View`, and `Out-the-Window`.

Optional Properties

Observer. Specifies the active observer.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvObserverMode",
  "Observer": "Observer 1",
  "ObserverMode": "XR"
}
```

3.5.7. MAK:VrvUseSavedView

The MAK:VrvUseSavedView message switches the observer to a saved view.

Required Properties

SavedViewFilename. Specifies a saved view file (*.osrx*) relative to the working directory of your VR-Vantage software (typically *C:/MAK/VR-Vantage x.x/bin64*).

SavedViewDisplayName. The name of a defined view in that file.

Optional Properties

Observer. Specifies the active observer.

SmoothTime. The amount of time in seconds to animate from the current view to the new view.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvUseSavedView",
  "Observer": "Observer 1",
  "SmoothTime": 3,
  "SavedViewFilename":
  "..\\appData\\settings\\stealth\\savedViewFile.osrx",
  "SavedViewDisplayName": "IED View"
}
```

3.5.8. MAK:VrvViewMagnification

The MAK:VrvViewMagnification message sets the view magnification for an observer in VR-Vantage. Useful for views that involve zooming, such as binoculars or cameras.

Required Properties

ViewMagnification. The amount to magnify the view.

Optional Properties

Observer. Specifies the active observer.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvViewMagnification",
  "Observer": "Observer 1",
  "ViewMagnification": 2.0
}
```

3.5.9. MAK:VrvSetSystemState

The MAK:VrvSetSystemState message enables or disables global display settings in a specific version of VR-Vantage. Each of these settings corresponds to a button in the Display Settings Toolbar of VR-Vantage. The settings are all optional. If any field is left out of this message, the value is set to false.

Required Properties

Vantage. Specifies the instance of VR-Vantage by the Site, App, and Exercise ID triple.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvSetSystemState",
  "Vantage": [ 2, 1, 1 ],
  "EntityLabels": true,
  "TacticalGraphicsLabels": true,
  "HeightAboveTerrainLines": true,
  "FireDetonateLines": true,
  "TacticalGraphics": true,
  "RadioCommunications": true,
  "VaporTrail": true,
  "PlanarReflections": true
}
```

3.5.10. MAK:VrvModelSet

The MAK:VrvModelSet message changes the model set in VR-Vantage.

Required Properties

Vantage. Specifies the instance of VR-Vantage by the Site, App, and Exercise ID triple.

ModelSet. The model set to use. The options are:

- 0 - 3D models.
- 1 - 2D icons.
- 5 - Colorized models (XR).

Optional Properties

ObserverMode. Specifies the observer mode: `Stealth`, `XR`, `Plan View`, and `Out-the-Window`.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvModelSet",
  "Vantage": [ 2, 1, 1 ],
  "ObserverMode": "Stealth",
  "ModelSet": 0
}
```

3.5.11. MAK:VrvModelScaling

The MAK:VrvModelScaling message enables and disables model scaling.

Required Properties

Vantage. Specifies the instance of VR-Vantage by the Site, App, and Exercise ID triple.

ModelScaling. Set to true or false.

Optional Properties

ObserverMode. Specifies the observer mode: `Stealth`, `XR`, `Plan View`, and `Out-the-Window`.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvModelScaling",
  "Vantage": [ 2, 1, 1 ],
  "ObserverMode": "Stealth",
  "ModelScaling": false
}
```

3.5.12. MAK:VrvModelScaleFactor

The MAK:VrvModelScaleFactor message sets the scale of the model if model scaling is enabled.

Required Properties

Vantage. Specifies the instance of VR-Vantage by the Site, App, and Exercise ID triple.

ModelScaleFactor. The amount to scale models.

Optional Properties

ObserverMode. Specifies the observer mode: `Stealth`, `XR`, `Plan View`, and `Out-the-Window`.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvModelScaleFactor",
  "Vantage": [ 2, 1, 1 ],
  "ObserverMode": "Stealth",
  "ModelScaleFactor": 50
}
```

3.5.13. MAK:VrvKeyPress

The MAK:VrvKeyPress message causes VR-Vantage to act as if a key has been pressed or released.

Required Properties

Key. Specifies the key. It follows the virtual key codes where 0 to 9 maps as 0x30 to 0x39 and a to z maps at 0x41 to 0x5A.

KeyState. Set to true if a key is pressed or released.

Optional Properties

Observer. Specifies the active observer.

Modifiers. Specifies if the **`Shift`**, **`Alt`**, or **`Ctrl`** keys are pressed.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvKeyPress",
  "Observer": "Observer 1",
  "Key": 41,
  "KeyState": true,
  "Modifiers": {
    "Shift": false,
    "Alt": false,
    "Ctrl": false
  }
}
```

3.5.14. MAK:VrvMouseEvent

The MAK:VrvMouseEvent messages causes VR-Vantage to respond as if a mouse event has occurred.

Required Properties

EventType. The type of mouse action to send. Supported event types are `Move`, `Wheel`, `ButtonPress`, `ButtonRelease`, and `ButtonDoubleClick`.

Optional Properties

Observer. Specifies the active observer.

X and Y. Specifies the current location of the mouse.

Wheel. Specifies the state of the mouse wheel.

Modifiers. Specifies if the **`Shift`**, **`Alt`**, or **`Ctrl`** keys are pressed.

Buttons. Specifies the current state of the mouse buttons.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvMouseEvent",
  "Observer": "Observer 1",
  "EventType": "Move",
  "X": 100.0,
  "Y": 100.0,
  "Wheel": 0.0,
  "Modifiers": {
    "Shift": false,
    "Alt": false,
    "Ctrl": false
  }
  "Buttons": {
    "Left": true,
    "Middle": false,
  "Right": false
  }
}
```

3.5.15. MAK:VrvTrackHistories

The MAK:VrvTrackHistories message enables or disables track histories in a specific instance of VR-Vantage.

Required Properties

Vantage. Specifies the instance of VR-Vantage by the Site, App, and Exercise ID triple.

TrackHistories. Specifies true or false.

Optional Properties

ObserverMode. Specifies the observer mode: `Stealth`, `XR`, `Plan View`, and `Out-the-Window`.

Example Message

```
{
  "InteractionType": "MAK:VrvTrackHistories",
  "Vantage": [2,1,1],
  "ObserverMode": "Stealth",
  "TrackHistories": false
}
```

3.5.16. MAK:VrvDynamicOcean

The MAK:VrvDynamicOcean message enables or disables the dynamic ocean in a specific instance of VR-Vantage.

Required Properties

Vantage. Specifies the instance of VR-Vantage by the Site, App, and Exercise ID triple.

DynamicOcean. Specifies true or false.

Optional Properties

ObserverMode. Specifies the observer mode: `Stealth`, `XR`, `Plan View`, and `Out-the-Window`.

Example Message

```
{
  "MessageKind": 2,
  "InteractionType": "MAK:VrvDynamicOcean",
  "Vantage": [2,1,1],
  "ObserverMode": "Stealth",
  "DynamicOcean": true
}
```

Index

E

F

G

H

I

J

K

L

M

N

O

P

W

X-Y-Z

Link - Simulate - Visualize

150 CAMBRIDGE PARK DRIVE, 3RD FLOOR CAMBRIDGE, MA 02140 617.876.8085 www.mak.com